



University of
Pittsburgh

Algorithms and Data Structures 2

CS 1501



Spring 2023

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines
 - Homework 2: this Friday @ 11:59 pm
 - Lab 1: next Tuesday @ 11:59 pm
 - Assignment 1: Friday Feb 17th @ 11:59 pm
- You can view correct answers for homework questions
- Jupyterhub server for testing out small snippets of code
 - jupyterhub.sci.pitt.edu
 - Use your Pitt username and password
 - Has to be either on campus or over Pitt VPN
 - You can connect to Pitt VPN using the PulseSecure program; instructions available on Pitt IT website
- TAs student support hours available on the syllabus page

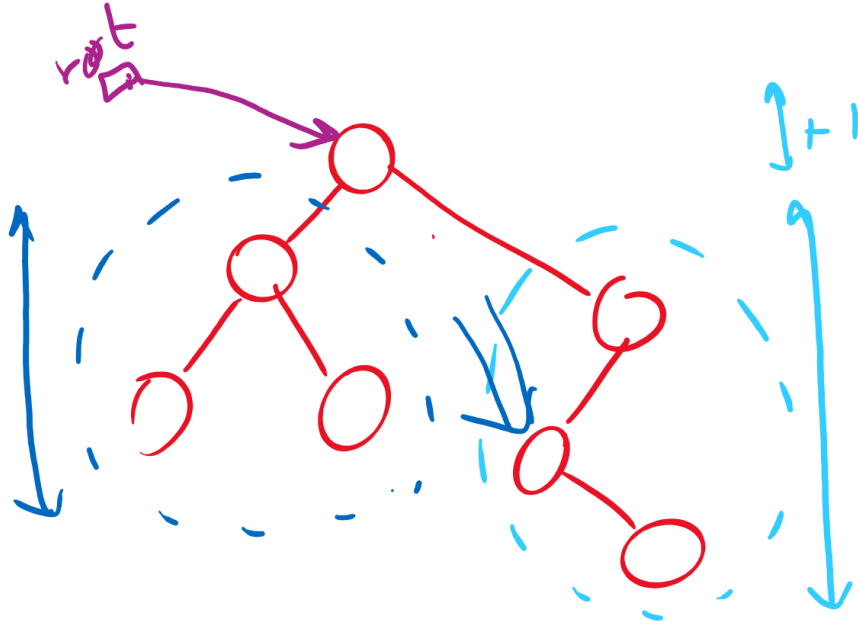
Previous lecture

- ADT Tree
 - Binary Tree
- BinaryTree implementation
 - BinaryNode

This Lecture

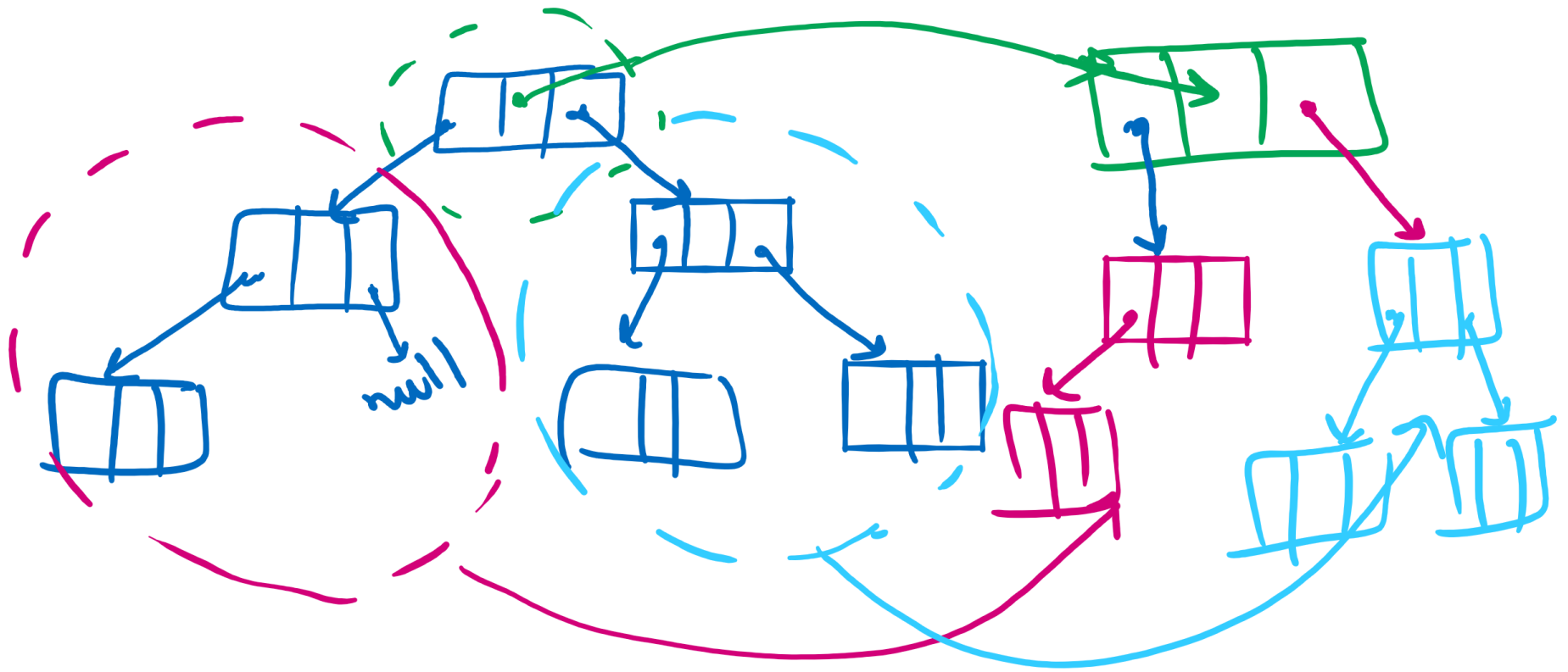
- Binary Tree Implementation
 - buildTree
 - tree traversals
- Binary Search Tree
 - How to add and delete
- Runtime of BST operations
 - Find, add, delete

Another implementation of getHeight



```
int getHeight(BinaryNode<T> root) {  
    int lHeight = 0;  
    int rHeight = 0;  
    if (root.left != null)  
        lHeight = getHeight(root.left);  
    if (root.right != null)  
        rHeight = getHeight(root.right);  
    return Math.max(lHeight, rHeight) + 1;  
}
```

BinaryNode.copy



buildTree method

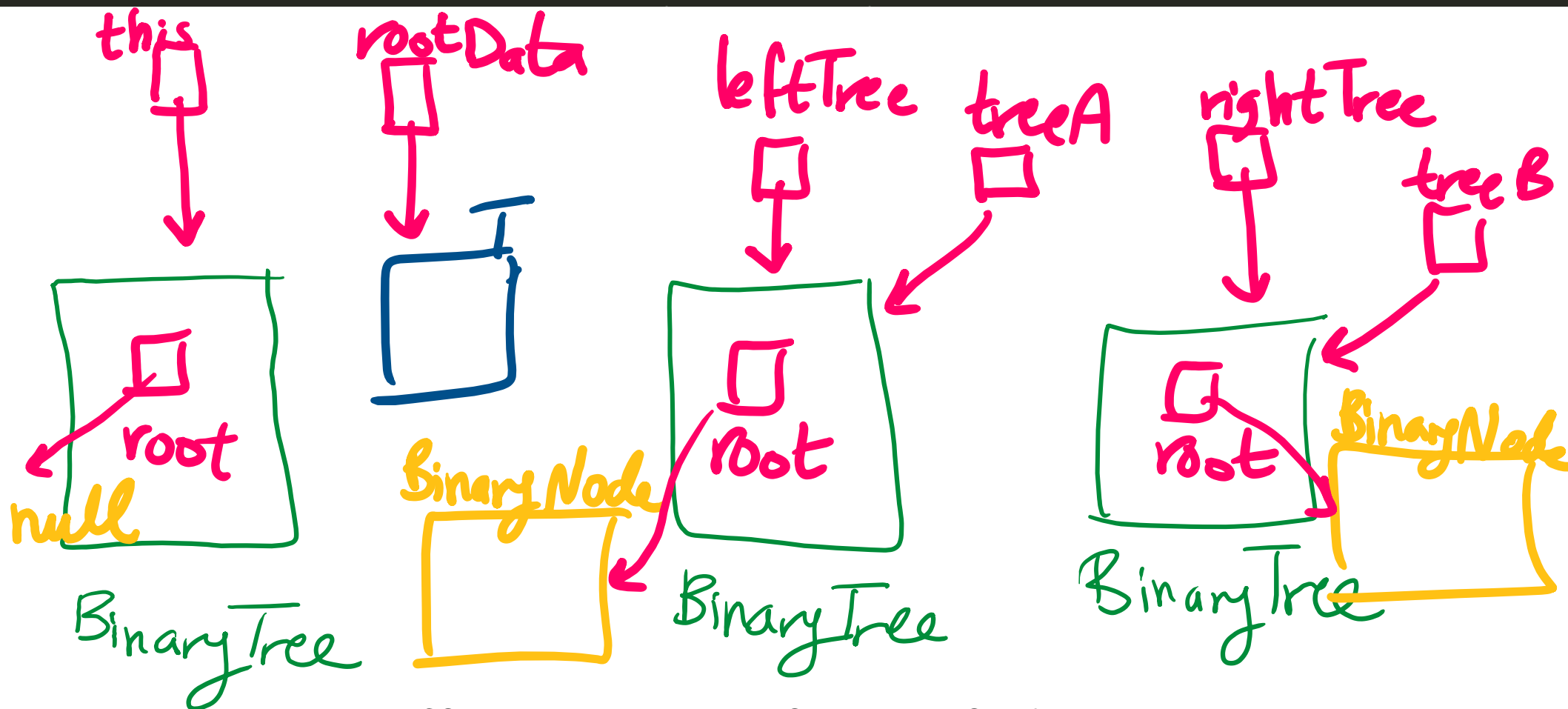
```
private void privateBuildTree(T rootData, BinaryTree<T> leftTree,  
                               BinaryTree<T> rightTree){
```

Let's draw a picture of the before state

- Given the call

```
privateBuildTree(data, treeA, treeB);
```

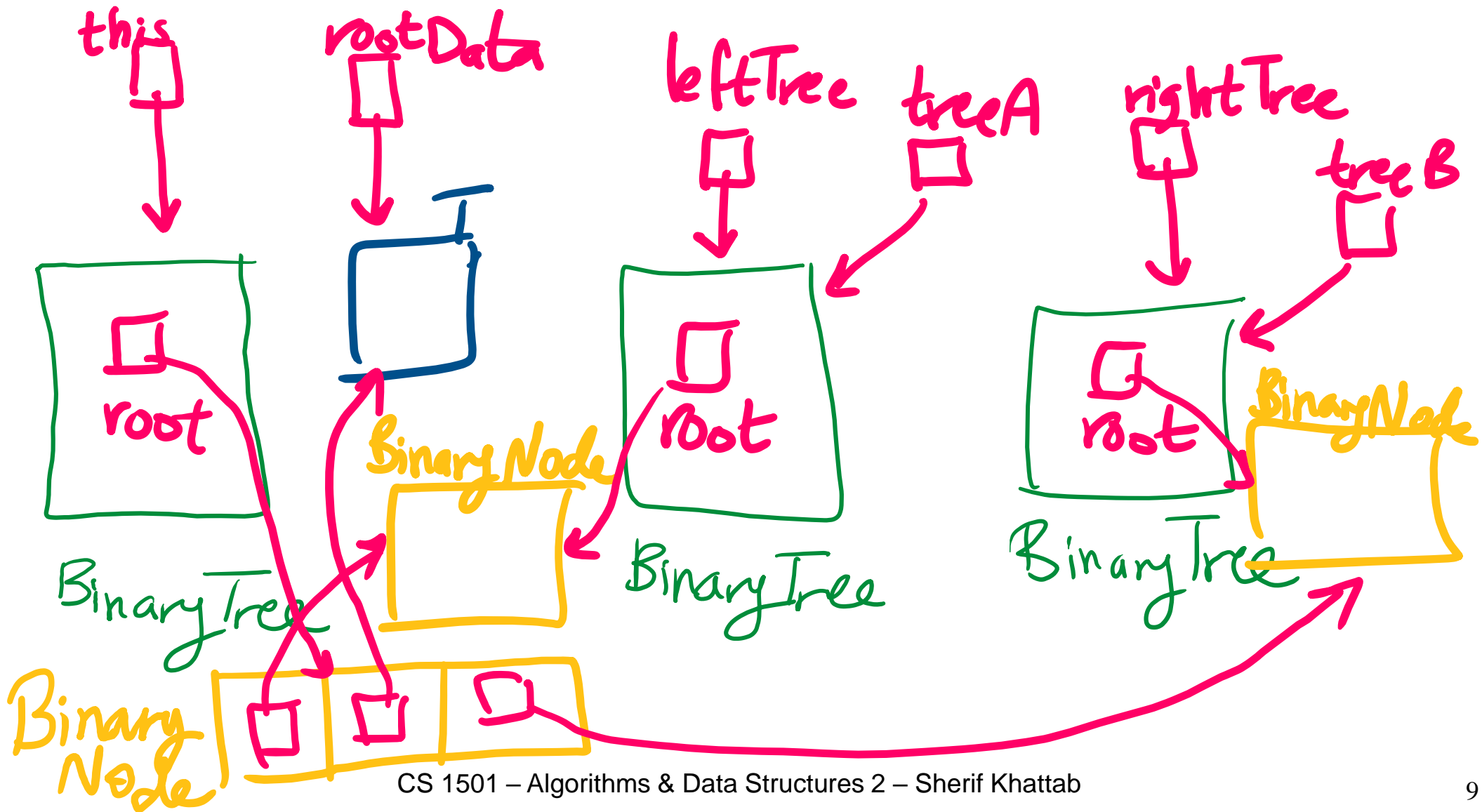
```
private void privateBuildTree(T rootData, BinaryTree<T> leftTree,  
                              BinaryTree<T> rightTree){
```



Let's draw a picture of the after state

```
privateBuildTree(data, treeA, treeB);
```

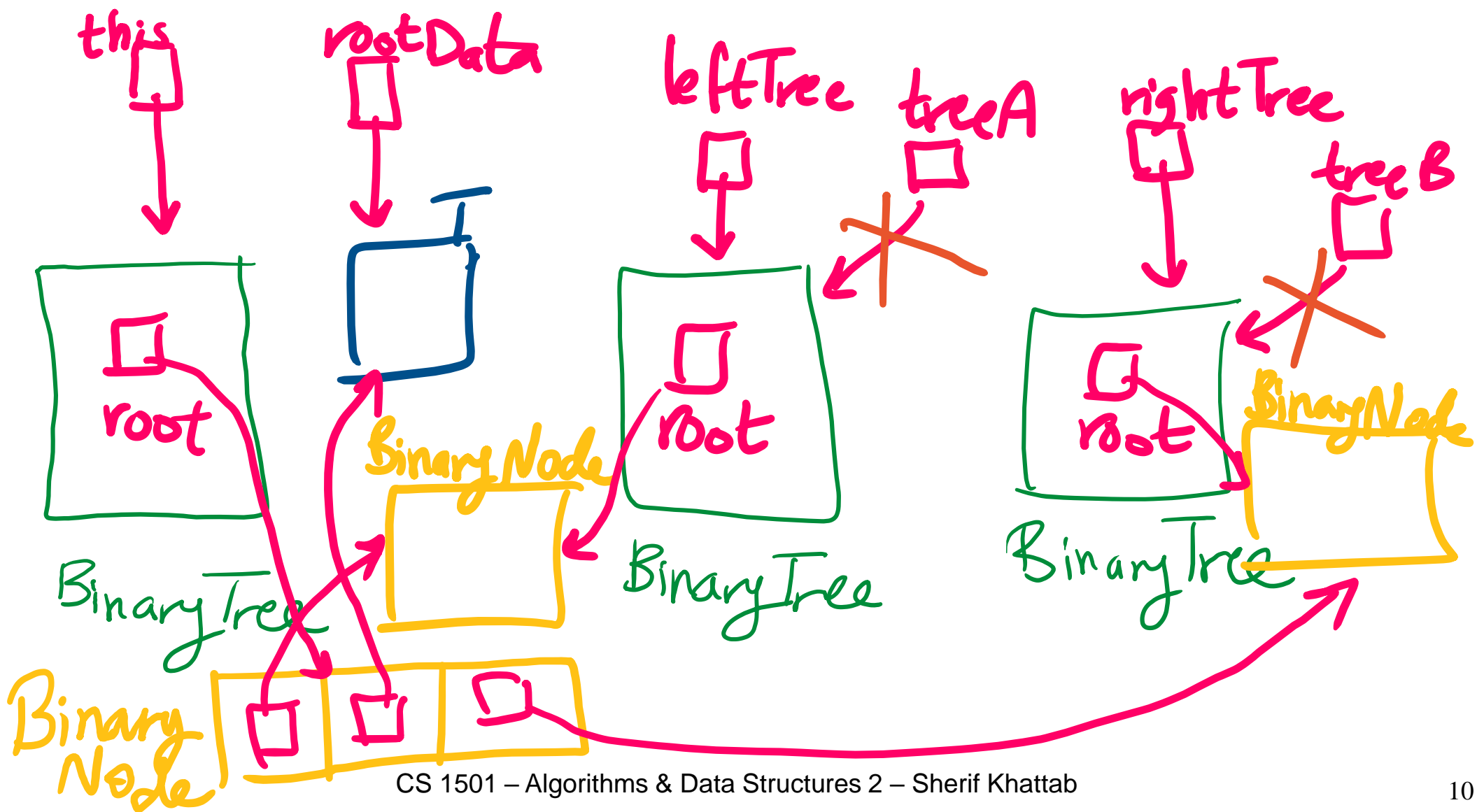
```
private void privateBuildTree(T rootData, BinaryTree<T> leftTree,  
    BinaryTree<T> rightTree){
```



Let's draw a picture of the after state

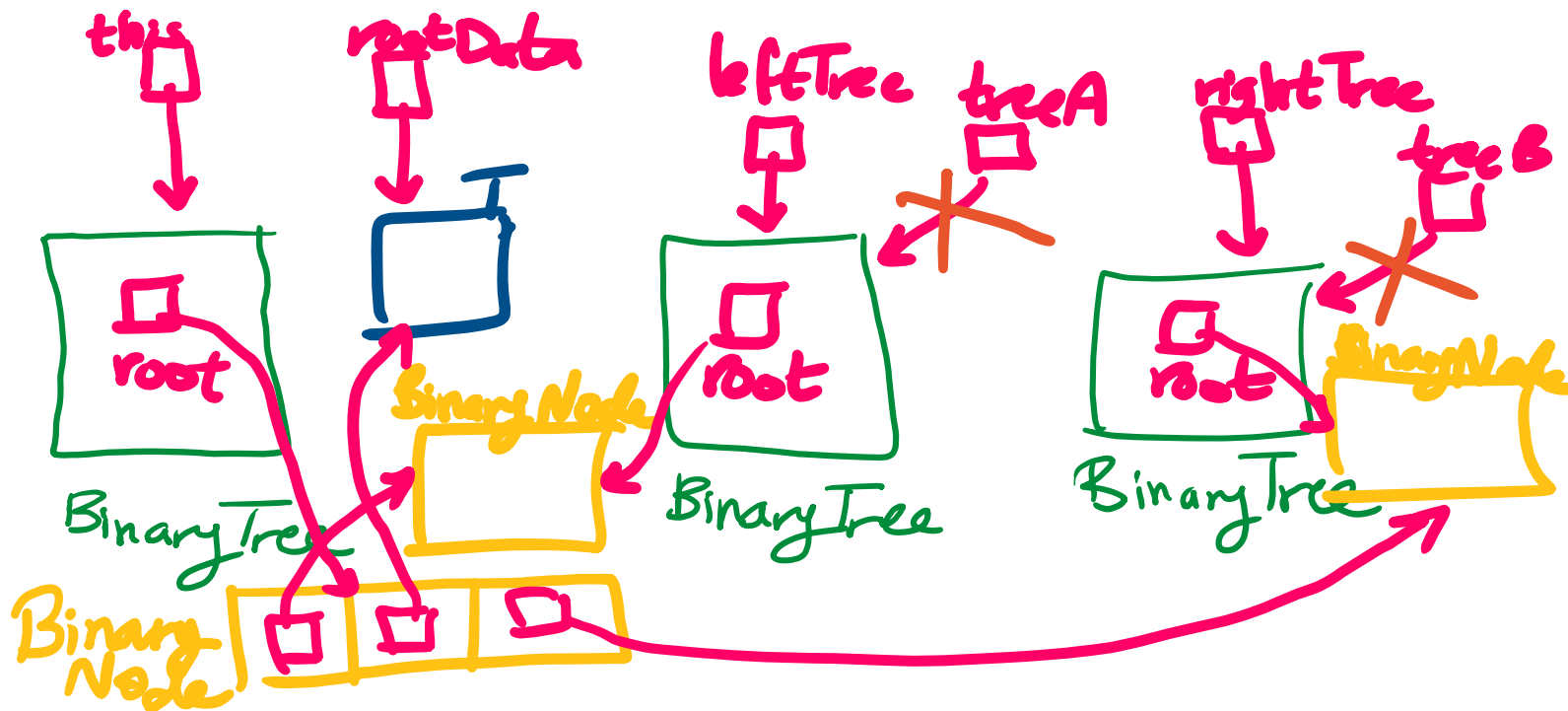
Need to also Prevent client direct access to this

treeA shouldn't have access this.root.left (same for treeB)



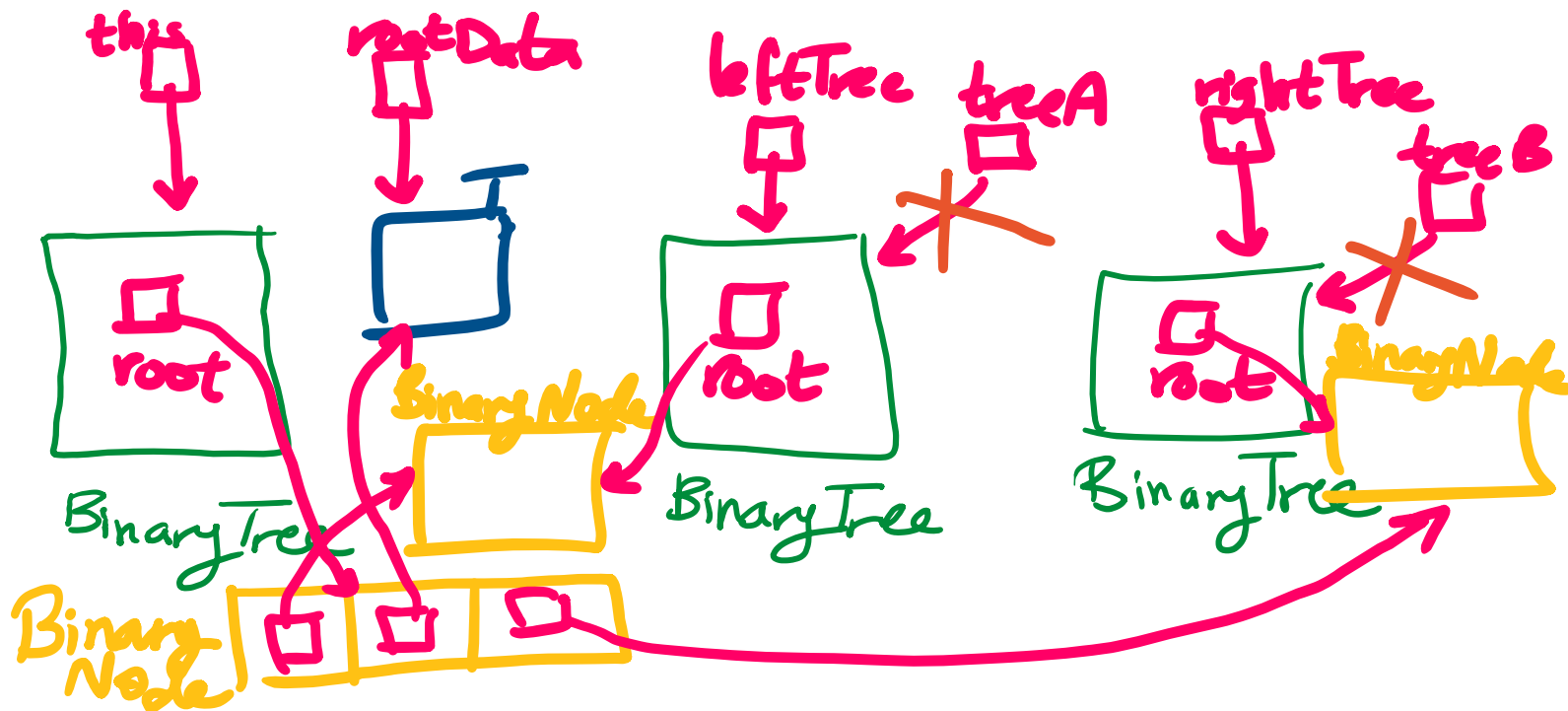
Main logic

- `root = new BinaryNode<>(rootData);`
- `root.left = leftTree.root;`
- `root.right = rightTree.root;`
- How to prevent client access?



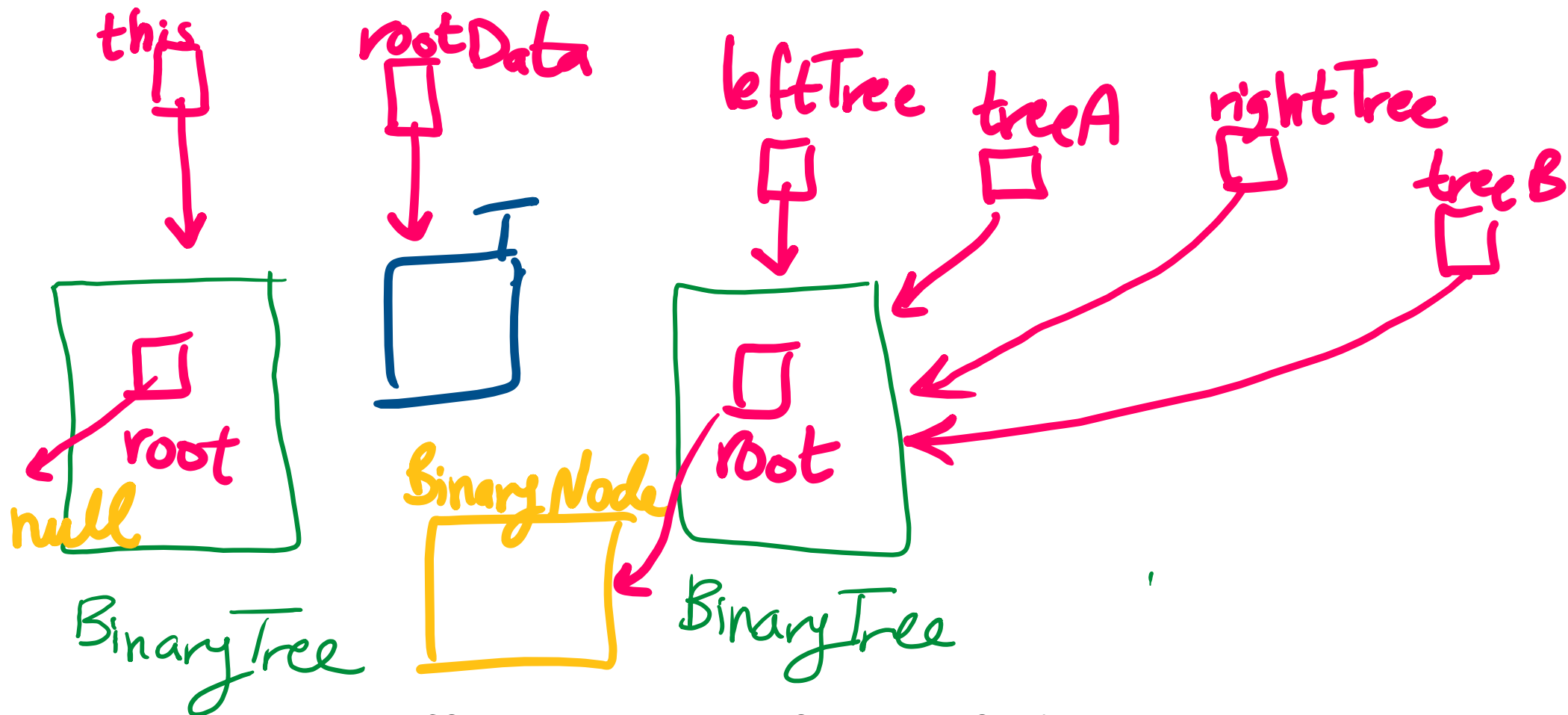
How to prevent client access?

- `treeA = treeB = null; //is that possible?`
- `leftTree = rightTree = null; //would that work?`
- `leftTree.root = null; rightTree.root = null;`



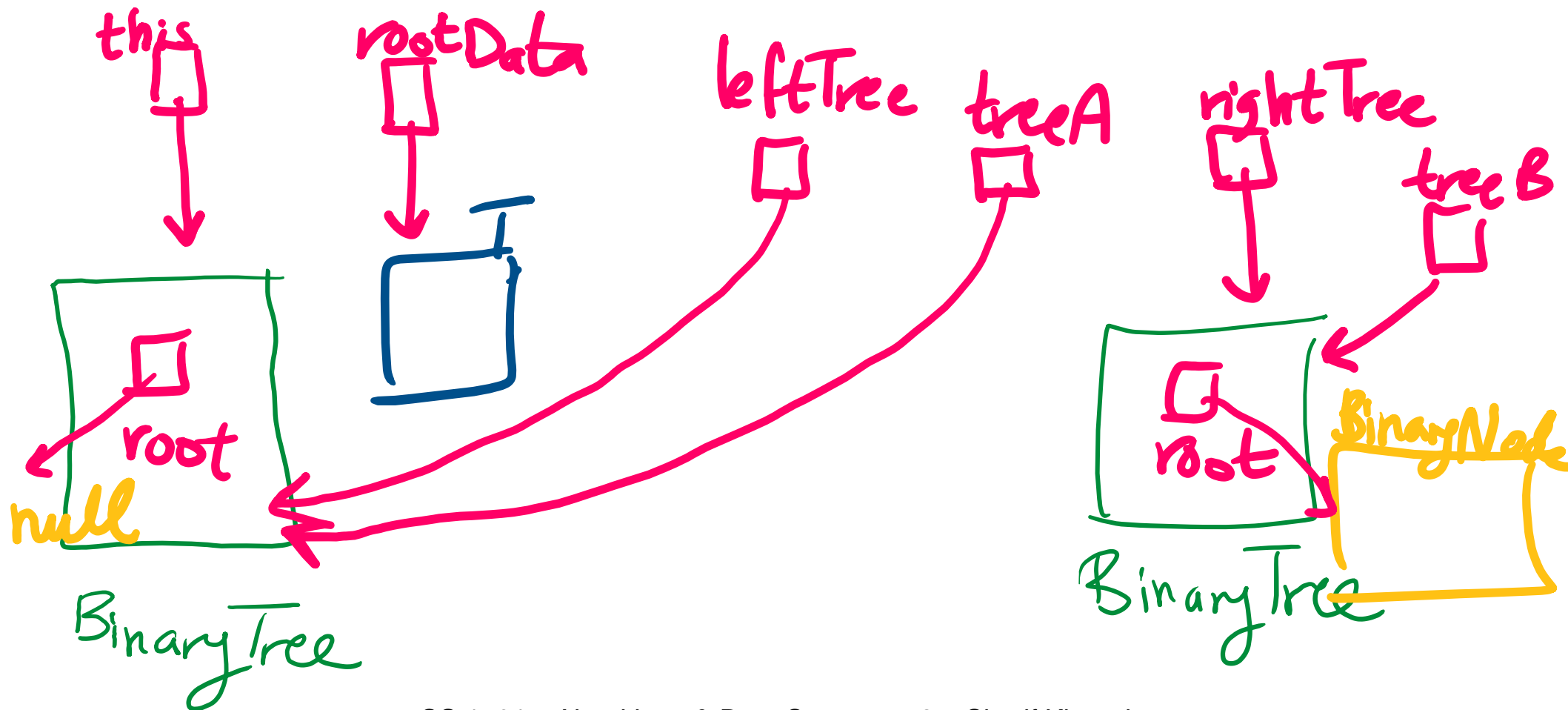
Special case: treeA == treeB

Need to make a copy of leftTree.root



Special case: treeA == this or treeB == this

Need to be careful before `leftTree.root = null` and `rightTree.root = null`



Tree Traversal Methods

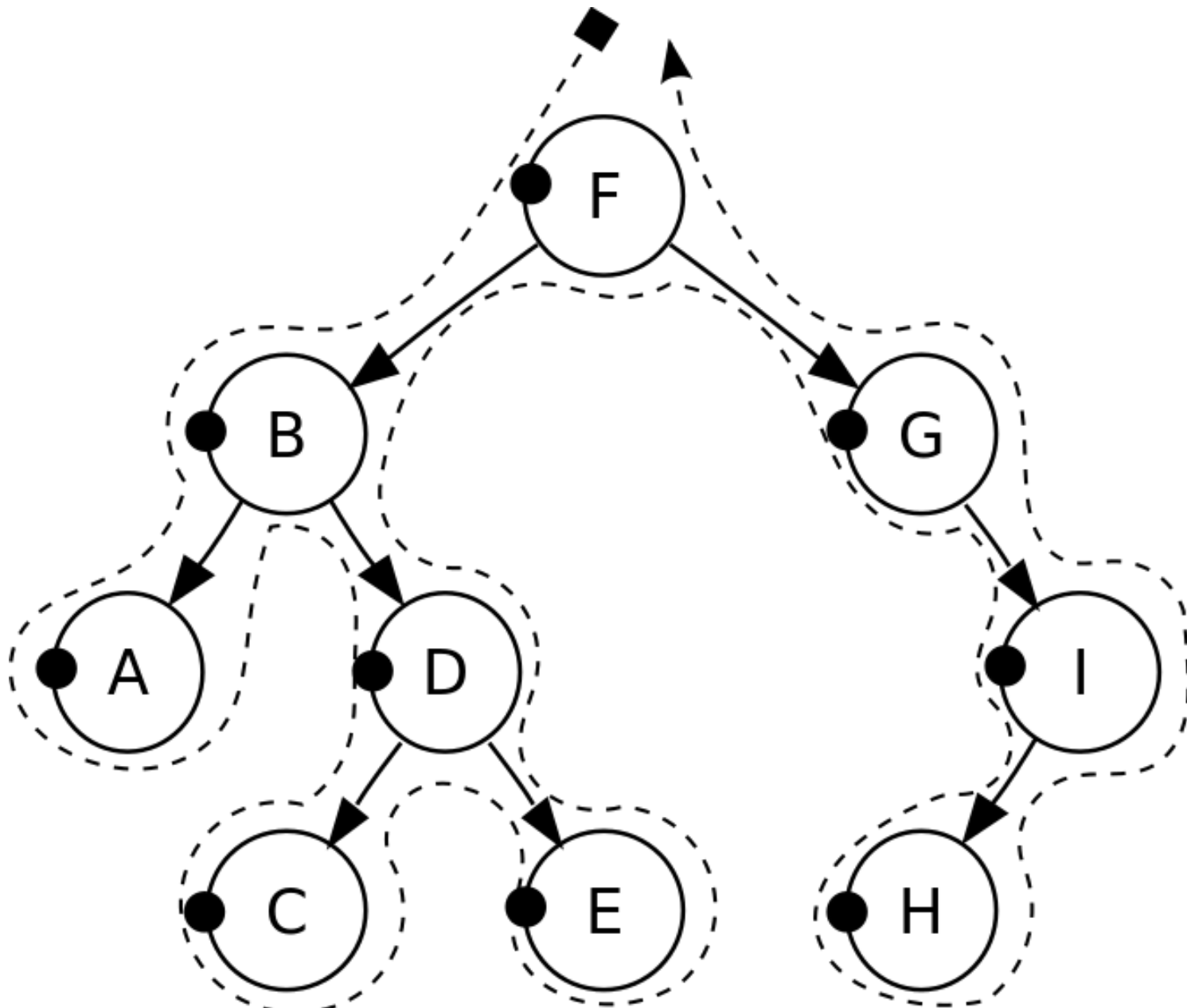
- How to traverse a Binary Tree
 - General Binary Tree
 - Pre-order, in-order, post-order, level-order

Traversals of a General Binary Tree

- Preorder traversal
 - Visit root **before** we visit root's subtree(s)

Pre-order traversal

F
B
A
D
C
E
G
I
H



Pre-order traversal implementation

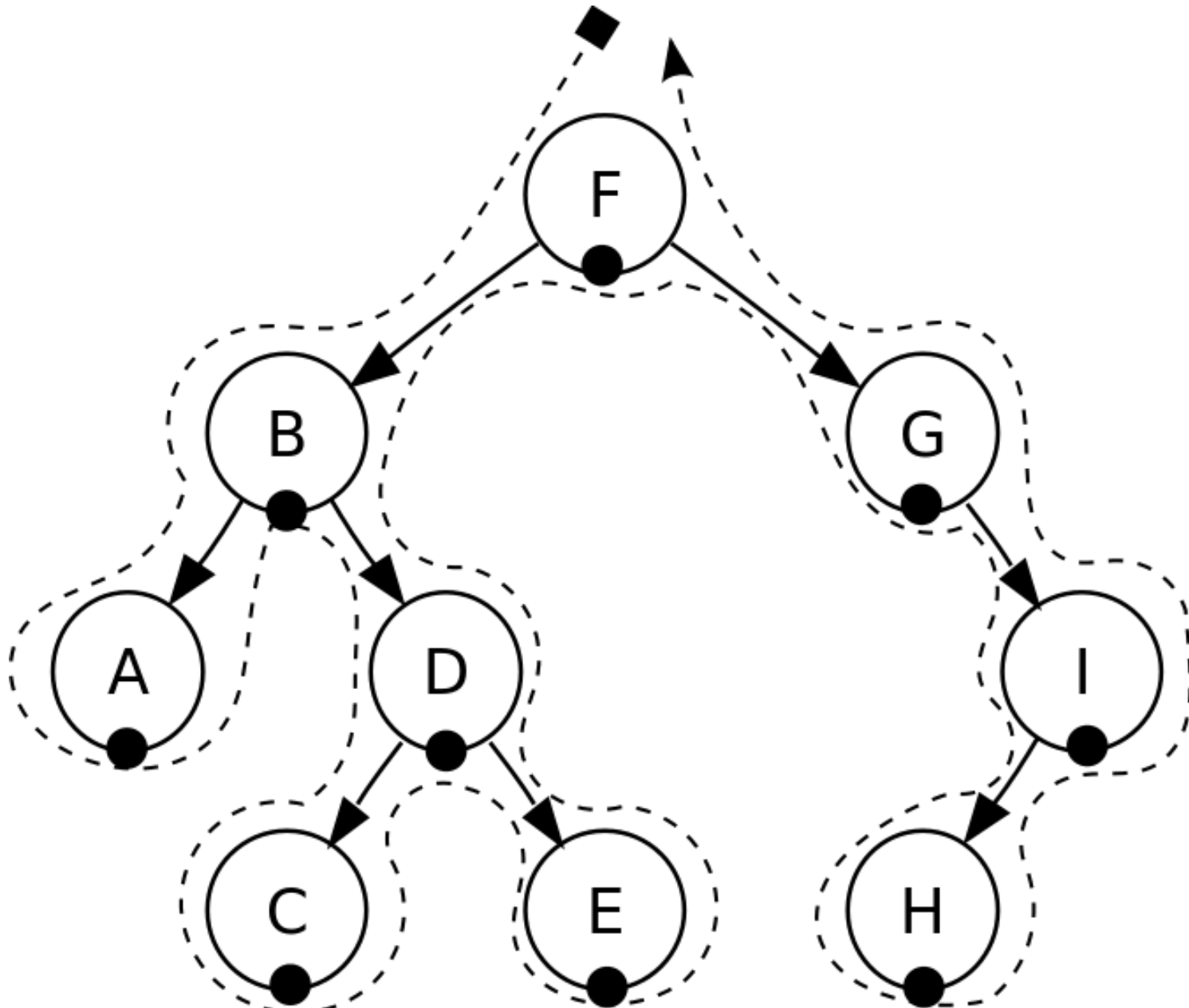
```
void traverse(BinaryNode<T> root) {  
    if (root != null) {  
        System.out.println(root.data);  
        traverse(root.left);  
        traverse(root.right);  
    }  
}
```

Traversals of a Binary Tree

- Preorder traversal
 - Visit root before we visit root's subtrees
- In-order traversal
 - Visit root of a binary tree **between** visiting nodes in root's subtrees.
 - left then root then right

In-order traversal

A
B
C
D
E
F
G
H
I



In-order traversal implementation

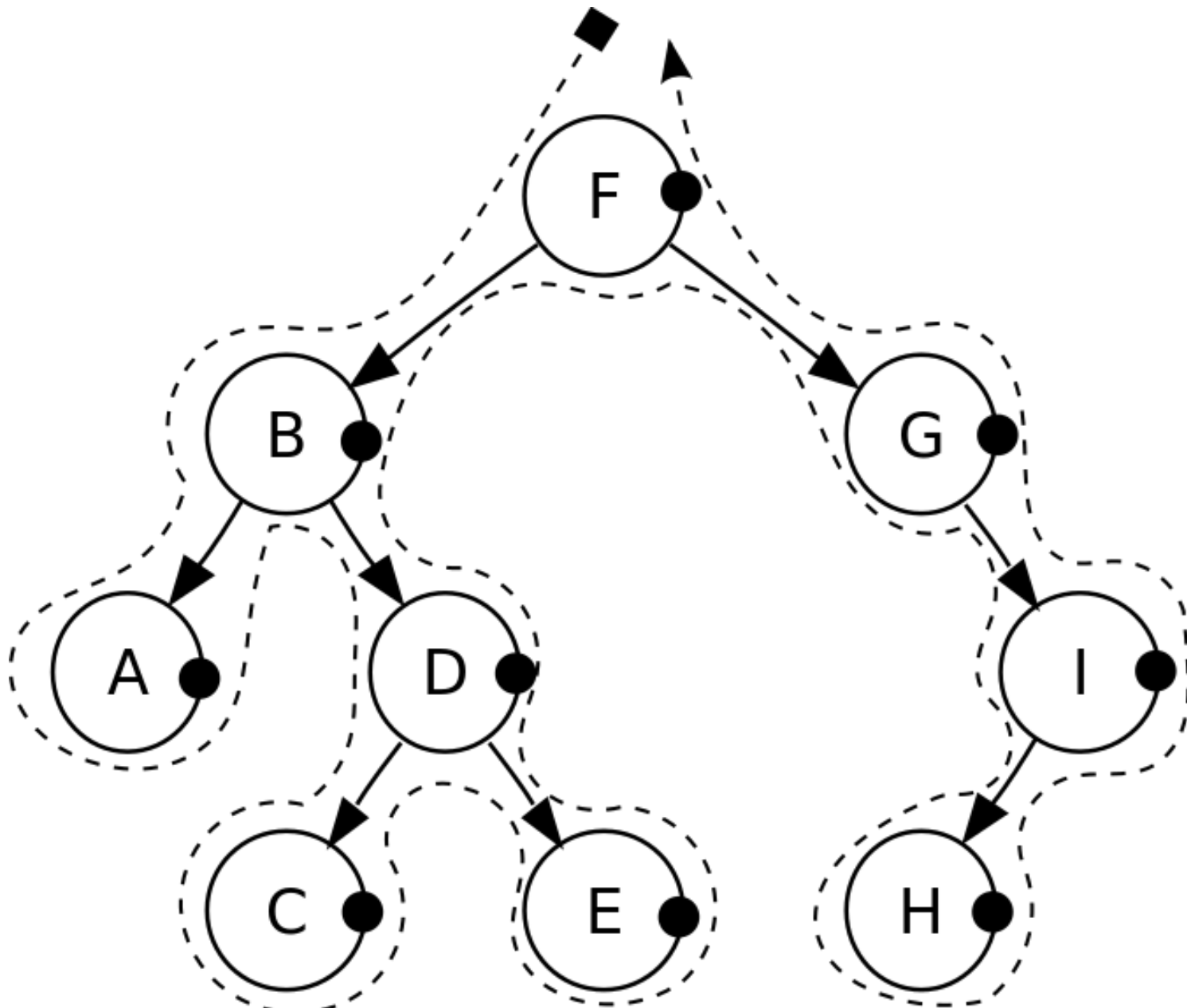
```
void traverse(BinaryNode<T> root) {  
    if (root != null) {  
        traverse(root.left);  
        System.out.println(root.data);  
        traverse(root.right);  
    }  
}
```

Traversals of a Binary Tree

- Preorder traversal
 - Visit root before we visit root's subtrees
- Inorder traversal
 - Visit root of a binary tree between visiting nodes in root's subtrees.
- Postorder traversal
 - Visit root of a binary tree after visiting nodes in root's subtrees

Post-order traversal

A
C
E
D
B
H
I
G
F



Post-order traversal implementation

```
void traverse(BinaryNode<T> root) {  
    if (root != null) {  
        traverse(root.left);  
        traverse(root.right);  
        System.out.println(root.data);  
    }  
}
```


Traversals of a Binary Tree

- Preorder traversal
 - Visit root before we visit root's subtrees
- Inorder traversal
 - Visit root of a binary tree between visiting nodes in root's subtrees.
- Postorder traversal
 - Visit root of a binary tree after visiting nodes in root's subtrees
- Level-order traversal
 - Begin at root and visit nodes one level at a time
 - We will see the implementation when we learn Breadth-First Search of Graphs

Tree Search Take 1

- *Traverse* every node of the tree
 - Is the key inside the node equal to the target *key*?
- How can we traverse the tree?

Tree Search Take 1

What is the runtime?

Can we do better?

Can we traverse the tree more intelligently?

Tree Search Take 2: Binary Search Tree

- Search Tree Property
 - For each node in the tree:
 - The data of the node is larger than the data in all nodes in the node's left subtree and
 - The data of the node is smaller than the data in all nodes in the node's right subtree

Let's build a Binary Search Tree

- Work in groups of 2-3 students
- Add the following integers to a Binary Search Tree in **the following order**:

10, 8, 17, 7, 5, 20, 15, 16, 4